# 6.869 Final Project: Image Style Transfer via CNNs and Texture Synthesis

Jun Wan
MIT
junwan@mit.edu

## Abstract

Style transfer is the process of migrating the style from a given image to another, creating a new image which is an artistic mixture of the two. There are two typical methods for handling the style transfer tasks. The first approach uses convolutional neural networks while the second approach takes an alternative path via generalization of texture synthesis algorithms.

In this paper, we explore both methods and implement style transfer using a various of different algorithms. We compare the results and propose novel improvement to improve efficiency. Our implementation is fast and flexible, being able to create visually pleasing results out of any pair of content + style images.

## 1. Introduction

Style transfer is the process of migrating a style from one image (the style image) to another (the content image). In Figure 1, we illustrate the style transfer task using the results obtained by our implementations. However, it is hard to define the "style" of a given image, not to mention migrating it to another image. Therefore, style transfer at its core is not well-defined and lacks clear mathematical definition. Specifically, style transfer is not accompanied by clear answers to questions such as:

- Which parts of the content image should be preserved or discarded?
- Which parts of the style image qualifies as "style" to be used?
- Should style transfer be allowed to distort key objects in the images like faces or buildings?
- What constitutes a successful style transfer result?

The uncertainty to these questions leads to two distinct approaches for handling the style transfer problem. The first approach uses convolutional neural networks, throwing these questions to the machines and try to generalize answers from the final CNN architectures. In the second approach, people first come up with their answers to these questions, then construct algorithms that match their belief. Interestingly, both approaches eventually draw their core intuition from texture synthesis. We shall discuss the previous work in details in the next section.

In this paper, we implement style transfer using both CNN based and texture synthesis based methods. For the CNN based methods, we apply pastiche optimization and feedforward using VGG19. For the texture synthesis based methods, we generalize the style transfer algorithms from the classical image-quilting algorithm [1] and texture synthesis methods [4]. The key idea is to divide the content image into overlapping blocks, find the satisfying style transfer result for each block and combine the results together. The results we obtain via our implementation are visually pleasing, containing rich and diverse parts brought from the style image, while keeping the essence of the content image intact.

The paper is organized as follows. In section 2, we introduce the related work, providing background for the methods we present in this paper. In section 3, we discuss our implementation using convolutional neural networks. In section 4, we discuss our implementation using image quilting and texture synthesis. Finally, in section 5, we conclude our results and discuss the potential future work. Due to the page limitation, we can not introduce every detail of our attempts. For things that we do not have pages for, we will enclose support material to illustrate them.

## 2. Previous Work

One of the earliest work related to style transfer is Efros and Freeman's paper [1] on texture synthesis. The paper suggested a patch-quilting procedure to synthesize texture, then generalized it to another closely related goal of texture transfer. To synthesize

Figure 1. The content image is on top. We provide six different style images and their corresponding style transfer results.

an image, they extract patches from the given texture, find the minimum cuts in the overlap regions between adjacent patches, and set the cuts as the boundaries between patches. Freeman later [2] replaced this patch merging procedure with a better one based on a Belief-Propagation (BP). BP serves as a combinatorial approximate solver for the energy minimization procedure, seeking the best patch assignments among a few neighbors per location so as to get the smoothest outcome. A direct follow-up of Freeman's work is Frigo et. al.'s style-transfer algorithm via texture-synthesis [3]. Unlike previous papers that used fixed-size patches and belief-propagation, Frigo et. al. divided the image into an adaptive quad-tree based on the distance to the nearest neighbor in the style image and the inner variance of the content patch. The results obtained by are far better than [1].

Another important work in texture synthesis is Kwatra et. al.'s approach [4] using the EM algorithm. Instead of the merging each patches individually, Kwatra et. al. adopted a global optimization point of view, seeking an overall synthesized image that would give the minimal accumulated local distance. This task has been broken into two stages. In the first stage, they fix the current global image and seek the best patch-matches. In the second stage, they fix the found matches and seek to update the global image aggregating all these chosen patches. This process is repeated until no improvement can be made. A follow-up of this work is Michael Elad and Peyman Milanfar's paper [5] in 2016, which generalize the approach to style transfer, creating results competitive with even the latest CNN style-transfer algorithms.

The first CNN paper on style transfer was proposed by Gatys et. al. [6] in 2015. They use a pre-trained CNN to extract features from both the style image and the content image and considered the style transfer problem as an energy-minimization task. Their ob-

jective is to find an image close to the content image, but also strongly correlated to the style image in the VGG feature domain. This method essentially reverses convolutional neural network, seeking an input image based on the features desired, and as such, leading to quite a demanding numerical optimization. The results provided in Gatys et. al.'s paper were very impressive and inspired a strong interest in this direction. But later work [7] mainly improves in speed. The main idea stays the same as [6].

## 3. Style Transfer via CNNs

In this section, we show how to implement style transfer using convolutional neural networks. Our first method is an optimization method. It does not train any neural network. Instead it uses a pre-trained VGG19 network to help extract features and optimize a pastiche (stylized image we wish to obtain) by minimizing the unified loss function.

The second method is a feedforward method. Given a style image, we train an ITN (image transformation network) that corresponds to that style. We feed the untrained ITN with a style image and the COCO dataset as content images. Following ITN is the same VGG19 network used before to compute the content loss and style loss. Then again we back-propagate the loss function, but this time to the ITN rather than the pastiche.

### 3.1. Opimization using pastiche

The pastiche is initialized randomly. It, along with the content and style images, are then passed through several layers of a VGG19 that is pre-trained on image classification. We use the outputs of various intermediate layers to compute two types of losses: style loss and content loss. These two losses represent how close the pastiche is to the style image in style and how close the pastiche is to the content image in content. We then minimized the losses by directly changing our pastiche image. After a few iterations, the pastiche image will gain both the style of the style image and the content of the content image. The equations for the style loss and content loss function are as below:

$$L_c = \sum_l \sum_{i,j} (\alpha C^l_{i,j} - \alpha P^l_{i,j})^2$$

$$L_s = \sum_l \sum_{i,j} (\beta G^{s,l}_{i,j} - \alpha G^{p,l}_{i,j})^2$$

$C$ represents the content image and $P$ is the output image. $G^{s,l}$ and $G^{p,l}$ are the gram matrixes for the style image and the output image, which are calculated based on the VGG19's feature domain. When implementing, we can tune the parameters $\alpha$ and $\beta$ to get better results. In Figure 2, we provide the style transfer result achieved when we apply our implementation on the Stata building.

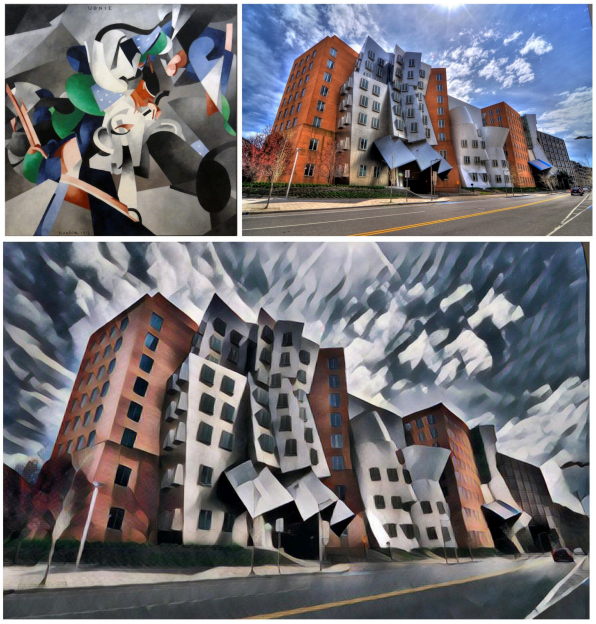

Figure 2. Example results of the pastiche optimization.

### 3.2. Feedforward method

Another way to handle the style transfer task is to create an untrained Image Transformation Network (ITN), which transforms the content image into its best guess at an appealing pastiche image. The pastiche image, along with the content and style images, is then passed through the same VGG19 network to compute the content and style losses. Finally, to minimize the loss, we back propagate into the parameters of the Image Transformation Network. We train this using the large COCO dataset. The implementation can transform any given picture into the style of some predefined artwork.

The transformation network we used is roughly the same as described in Johnson et al. [7], with the exception of using Ulyanov's instance normalization instead of batch normalization. The loss functions we use are close to the one described in [6]. The only difference is that we use VGG19 instead of VGG16. We also use "shallower" layers (e.g. we use relu1_1 rather than relu1_2) that differs from Johnson's implementation. This results in larger scale style features in transformations. In Figure 3, we provide the results we achieve using the feedforward method.

Figure 3. Example results of the feedforward method.

### 3.3. Conclusion

The two methods produced visually pleasant images. Through experiments, we found that the artistic style rendered places that have high contrast pretty well, but had a difficult time figuring out what to do with places that have low contrast, such as sky and water surface. Another large defect on this method is running time. For the optimization method, it takes 10 hours on an Amazon EC2 p2.x instance to produce an image. For the feedforward method, it takes 8 hours to produce an ITN for one style image, although the style transfer for a content image is fast (less than 1s).

## 4. Style Transfer via Texture Synthesis

Texture synthesis is the task of creating a large texture image out of a small input texture. A successful texture synthesis algorithm should be able to take any sample of texture and generate an unlimited amount of image data which, while not exactly like the original, will be perceived by humans to be the same texture (definition in [1]). In this section, we introduce the texture synthesis algorithms we implement and show how to turn them into style transfer algorithms.

### 4.1. Freeman's quilting algorithm

We characterize the key steps of Freeman's image quilting algorithm as below. First, we divide the image to be synthesized into overlapping square blocks and go through the blocks in raster scan order. For each block $B_i$, suppose $B_i$ is overlapped with $B_j$ and $B_k$, we search the input texture for a sub-square $I$ (of the same size as $B_i$) that agrees with $B_j$ and $B_k$ the most on the overlapped regions. Finally, we compute the error surface between the newly chosen block $B_i$ and the old blocks at the overlap region, and find a minimum cost path that completely divides $B_i$ with its neighbors. We set this path as the boundary between $B_i$ and neighboring blocks.

When we implement Freeman's algorithm, we encounter two technical issues. The first one is how to find a sub-square in the input texture that is closest to a given block $B$. Suppose the input texture's size is $h \times w$ and the block size is $b \times b$, then calculating with brute force requires $\Theta(h*w*b*b)$ time. This is totally unacceptable, not to mention we have to do such an operation hundreds of times. To solve this problem, we find Elkan's paper on k-mean clustering [8], which significantly boost our code speed. We will introduce Elkan's algorithm in details later.

The second issue is how to find the minimum cost path that divides a block from its neighbors. The task is easy if a block only has one neighbor, in which case we can just use a variation of the Dijkstra's algorithm. But with multiple blocks, things are not that simple. Eventually, we decide to find the cuts between a block and each of its neighbor. We then combine the cuts at the point where they intersect and greedily construct a cut that divides a block from all of its neighbors.

In Figure 4, we provide examples of successful and failed image quilting. In the first example, there are barely visible boundaries in the synthesized output and

the result is overall satisfying. However, in the second example, there are some visible boundaries. We think this is because the input texture is inconsistent in shade, thus sometimes it is hard to find a satisfying block that is close to all of its neighbors.

Figure 4. The first image is a successful example texture synthesis, while the second image is a failed one. Left: input texture images. Right: the synthesized results.

Finally, to extend the image quilting algorithm to style transfer, we need to modify the error term of the image quilting algorithm to be the weighted sum $\alpha$, times the block overlap matching error plus $(1-\alpha)$ times the squared error between the correspondence map pixels within the source texture block and those at the current target image position. The parameter $\alpha$ determines the tradeoff between the texture synthesis and how much we want to preserve the content in the content image. This is repeated for 5 rounds, reducing the block's size and increasing the value of $\alpha$ in each round. As suggested in [1], we set $\alpha = 0.2 * (i-1) + 0.1$ in the $i^{th}$ round. We provide our output result in Figure 5.

### 4.2. Texture synthesis using EM

The second method is similar to energy minimization. Let $X$ denote the texture over which we want to compute the texture energy and $Z$ denote the input sample to be used as reference. Again, we divide the image $X$ into blocks, but this time the blocks are strongly overlapped. Each pixel in $X$ is covered by at least 8 blocks. We denote the set of blocks as $B$ and let $X_b$ denote the corresponding area on $X$ for any $b \in B$. The objective is to find $X$ and the set $\{Z_b \mid b \in B\}$ such that

$$E(X, \{Z_b\}) = \sum_{b \in B} ||X_b - Z_b||^r,$$

is minimized. Here, $Z_b$ can be any sub-square in $Z$ and $r$ is a parameter that we set as 0.8. Kwatra et. al.'s approach is simple. We first fix $X$ and optimize on the set $\{Z_b\}$. We then fix $\{Z_b\}$ and find the optimal $X$. This is repeated until no further improvement can be made. We successfully implement their algorithm with Matlab. Due to page limitation, the results are enclosed as support material.

Figure 5. Style transfer result using image quilting. Left: content image. Middle: style image. Right: the style transfer result.

To extend this method to style transfer, we need to modify the error term, adding a new term that guarantees $X$ is close enough the content image. Other than that, the algorithm stays the same. Unfortunately, our implementation of style transfer using this method is not successful. The results look very blurry. We think maybe this is because each pixel is covered by too many blocks, and that we need more rounds with different block sizes and weights. But we did not have enough time to fix this problem.

### 4.3. Boosting speed using k-mean clustering

In both the quilting and EM approaches, a key obstacle is to find a sub-square in the image that is closest to a given block $B$. Among both algorithms, this step takes the longest and solely determines the algorithms' run time. We now show how to do this fast.

We use hierarchical clustering to organize the input neighborhoods into a tree structure. Initially, we only have the root node which contains all the possible

sub-squares in the image. Starting at the root node, we perform the k-means clustering (we set $k = 4$) in [8] over all sub-squares in that node. We then create k children nodes corresponding to the k clusters and apply the above step recursively to each of the children nodes. This is continued until the number of sub-squares in a node is less than 1% of the total number of sub-squares. Fortunately, for each sub-square, we only have to save its location in the image and the corresponding sub-vectors are constructed on the fly.

The k-mean clustering algorithm in [8] exploits triangle inequality to avoid unnecessary computation. The key intuition is that for any point $x$ and centers $c, c'$, if $d(c, c') \geq 2d(x, c)$ then $d(x, c') > d(x, c)$. By constantly checking the triangle inequality, we can reduce the number of distance computation from $O(n \cdot k \cdot r)$ to $O(n)$, where $n$ is the number of samples, $k$ is the number of centers and $r$ is the number of rounds. This is a huge improvement, after we apply k-mean clustering, the run time decreases from 1 hours to approximately 5 minutes.

### 4.4. Other methods

Beside the quilting and EM approaches, we also tried to implement the split-and-match algorithm in [3]. We successfully implemented the split part using C++. But unfortunately, in the match part, the paper does not thoroughly explain some of its key techniques but points to other papers for reference instead. We did not really understand how to apply those techniques into the match part. Never the less, this is helpful because we can use the adaptive quad tree achieved in the split part and combine it with the quilting or texture synthesis algorithm. Due to page limitation, we do not discuss the details here. We will enclose an example output of the splitting algorithm as support material.

## 5. Conclusion

In this paper, we explore the style transfer task from both CNN and texture synthesis perspective. Our implementations using CNNs produce better results but the texture synthesis based methods give us more insights into the key question "what constitutes a successful style transfer". We think that for the content image, features like edge and local contrast should be preserved while colors and shades can be altered as will. For the style image, any part should be considered as part of the "style". Another huge difference between the two methods is that texture synthesis based algorithm essentially merges patches from the style image. But CNN-based methods are much more likely to synthesize unseen structures. Maybe this is why CNN-based methods achieve a better performance.

In term of speed, texture synthesis based algorithms are much more faster. However, for a fixed style image, CNN-based method can pre-train the neural network so that given any content image, it can output the transfer result in less than 1 second.

## 6. Personal Contribution

This is a joint project with my teammate Chenxing Zhang. My personal contribution is listed as follows:

- Participate in the paper survey of both CNN based methods and texture synthesis based methods.
- Implement the image quilting algorithm and the EM texture synthesis algorithm using Matlab.
- Implement the split part of the split-and-match algorithm using C++.
- Write half of the report.

## References


[1] Alexei Efros and William Freeman, Image Quilting for Texture Synthesis and Transfer. Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 341–346, 2001.

[2] William Freeman, Thouis Jones, and Egon Pasztor, Example Based Super-Resolution. IEEE Computer Graphics and Applications, Vol. 22, No. 2, pp. 56–65, 2002.

[3] Oriel Frigo, Neus Sabater, Julie Delon and Pierre Hellier, Split and Match: Example-Based Adaptive Patch Sampling for Unsupervised Style Transfer, CVPR, 2016.

[4] Vivek Kwatra, Irfan Essa, Aaron Bobick and Nipun Kwatra, Texture Optimization for Example-based Synthesis. ACM Trans. Graph., pp. 795–802, 2005.

[5] Michael Elad and Peyman Milanfar, Style-Transfer via Texture-Synthesis. Trans. Img. Proc., pp. 2338–2351, 2017.

[6] L.A. Gatys, A.S. Ecker, and M. Bethge, Texture Synthesis Using Convolutional Neural Networks, NIPS 2015.

[7] J. Johnson, A. Alahi, and F.-F. Li, Perceptual Losses for Real-Time Style Transfer and Superresolution, ECCV 2016.

[8] Charles Elkan, Using the Triangle Inequality to Accelerate k-means. Proceedings of the Twentieth International Conference on International Conference on Machine Learning, pp. 147–153, 2003.